recommendations for salvaging puppy linux

figosdev / version 0.1 / august 2019

introduction

years ago, in the era of pizzapup and puppy 2.x, i learned more about gnu/linux from puppy than any other distro. ultimately i would move on to dsl and xubuntu because of "driver support." i was a refugee from dos and windows (dos first) and didnt really get the modules / kernel business. i was good with computers, but a noob with gnu/linux.

ive tried more than one generation of puppy, and i dont think the new stuff is all bad. even though i was hesitant to move from puppy with the 2.x kernel and newer versions seemed slower on the same old hardware (i think i wanted it all to work on a pentium ii and puppy was fast!) i still wanted wifi to work. those ndiswrapper scripts and zdrv-like hacks did not have defaults that helped me as a noob.

ive spent a lot of time taking puppy apart and putting it back together. some of the things that matter the most to you really dont matter to me-- and some of the things that matter to me really dont matter to you.

puppy is probably the most endlessly customised (certainly remixed and reinvented) distro that ever existed.

but it was very noob-friendly and very lightweight, and really put way too much power in the hands of people (like me) that didnt know what theyre doing.

that is its best feature, and that is what i find missing from the puppy community-- it really lacks the spirit of those days.

there are far too many changes to puppy to salvage all of it, and far

too many different ideas to make everyone happy, and far too few developers to make the changes needed to make everyone happy.

ive stayed in touch with people from the community over the years, and i talk to them other places as well. there are people ive spoken to since they left the puppy forum. ive joined the developer community (and left) trying to fully understand what theyre doing.

my approach to puppy these days is more in the capacity of an investigative reporter, but as i said ive spent a lot of time taking puppy apart and putting it back together-- and ive done that with my own code.

how to be technical when youre not already technical

puppy has always sought to be friendly, and its model of friendliness is built on jwm, rox-filer and the desktop (icons) provided by rox-filer. personally i have preferred pcmanfm (or spacefm, though i do like pcmanfm a little bit better and pcmanfm-qt is alright) for ages, but with good default settings rox-filer is quite tolerable. we dont have to agree on the default settings.

puppy users pick up technical terms on a regular basis: mount, iso, term window, sfs, remaster, bash, kernel-- puppy worked before i understood these things, but people who want to learn more are rewarded for it.

there is, perhaps, some line between a highly-technical user and a developer. a lot of it is about commitment. some of it is about technical details.

this book is made for both developers and people who want to make a real difference regarding the future of puppy. it will require both. you are encouraged to read this, and **if you find something technical-- feel free to skip it.** try to get what you can from it, and just keep going. just like a bullwinkle cartoon has jokes for the kids and other jokes for the grownups, this has advice for more than one group. take

whatever you can from it-- ignore the rest and dont get hung up on it.

for example, if we talk about basic steps in remastering, its good enough to know what remastering is-- bonus points if you care what the steps are. bigger bonus points if you care about the tools mentioned. if you dont know what remastering is, just skim the words until the subject has moved onto something

else. if you just cant help yourself, write down any concepts you want to look up later.

a lot of people become technical gradually, comfortably, not all at once. puppy is great in that regard, because you can go from booting a usb to helping a fellow user, to creating packages and maintaing a pup of your own, if you want to.

one of the reasons that puppy made it so simple to do that, is that puppy was simple-- it wasnt just designed to be simple on the outside, it was designed without a lot of technical considerations that are expected of a larger distro like debian or (going back a bit) mandrake.

it would take you ages to build a distro from source, but to start tinkering and gradually learn more and more of what it takes to customise your own distro can be very rewarding. and the most ideal future of the gnu/linux distro is one that is more customisable than ever. it will need to be, just to save gnu/linux from the threats aimed at it.

remember windows?

ive used windows 1.01, 3.0, 3.1, 95, 98, me, 2k, xp, 7, 8 and 10.

ive used dos 3, 5, 6 and (as it comes with windows 95) 7.

it used to matter a lot more that puppy was familiar to windows users. it still matters (though less.)

what about mac users? i mean, you can put wbar (or one of the fancier equivalents-- wbar is what tinycore has promoted) on the desktop in place of a taskbar, or put the taskbar at the top.

dos is very much worth mentioning. i hope youll stick around for this one, even if you feel a need to skip every other word.

dos has a lesson for the future of gnu/linux: **the boot disk.**

because a dos boot disk didnt have ROOM for too much complexity. sure, parts of it were stupidly technical, but there were loads of simplicity, because it was impossible to do more:

text screen: graphical programs existed. to open them from a modern shell, you find the icon on the screen and click on it. on a dos system, you said **dir** and then typed the name of the program you wanted to run.

shortcuts: windows link files didnt exist on early versions of dos. to create a friendly shortcut like **paint** for something ugly like **pbrush /m ps2 /r 320x200** you (or someone who helped you) would put that text in a file called **paint.bat** and now to run all that, you just need to type **paint** and hit enter.

no icon was included-- though graphical shells did exist in dos (they were kind of wasteful on machines without much ram.)

floppies: sort of the usb drives of yesteryear-- i said in the late 1980s or early 90s that we would be better off storing files on chips. i wasnt sure it would happen, though i was sure that they would be easier to carry.

finding files: a floppy held very little data, so you didnt need to bother searching through folders to find your file-- just run the dir command. i used a program called ddir, which had output similar to ls -l except in two columns. ls is multicolumn by default, but ls -l is not. this is the closest thing i have to ddir for puppy, i like it:

```
root:/home/anon/Desktop/master#> find | dc 113
. ------------------------------------------------------------------------------------------------------------------
fig-master       4096 2019-07-06 07:23:10  unzip          173796 2019-07-06 07:28:56
fig-master.zip 857268 2019-07-06 07:28:39
./fig-master ---------------------------------------------------------------------------------------------------
examples                   4096 2019-07-06 07:23:10  LICENSE                    6970 2019-07-06 07:23:10
cs.fig                      189 2019-07-06 07:23:10  README.md                   199 2019-07-06 07:23:10
figbook.pdf              162303 2019-07-06 07:23:10  from_examples_folder.pdf 383949 2019-07-06 07:23:10
C3ekoCR.png                6330 2019-07-06 07:23:10  fig46.py                  59828 2019-07-06 07:23:10
./fig-master/examples ------------------------------------------------------------------------------------------
LICENSE                      6970 2019-07-06 07:23:10  arrdo.fig                    81 2019-07-06 07:23:10
arrlen.fig                   1761 2019-07-06 07:23:10  arrname.fig                1372 2019-07-06 07:23:10
asciimint.fig                 778 2019-07-06 07:23:10  bestwithpygameinstalled.fig  665 2019-07-06 07:23:10
cf.fig                       1006 2019-07-06 07:23:10  colourfind.fig             2985 2019-07-06 07:23:10
cubes.fig                    1819 2019-07-06 07:23:10  dircmp.fig                 1079 2019-07-06 07:23:10
egrepv.fig                    339 2019-07-06 07:23:10  figsp.fig                  1051 2019-07-06 07:23:10
figsph.fig                    988 2019-07-06 07:23:10  findsim.fig                3261 2019-07-06 07:23:10
floral.fig                   1842 2019-07-06 07:23:10  gemseven.fig               1106 2019-07-06 07:23:10
installcolorama.fig            87 2019-07-06 07:23:10  nodup.fig                   147 2019-07-06 07:23:10
recur.fig                     156 2019-07-06 07:23:10  rgb.fig                     519 2019-07-06 07:23:10
showtwo.fig                   485 2019-07-06 07:23:10  sp.fig                      782 2019-07-06 07:23:10
fige.pdf                   383949 2019-07-06 07:23:10
root:/home/anon/Desktop/master#>
```

rox-filer can do this with a gui.

the way you added "**startup programs**" was to name them in a file called autoexec.bat. in debian it was (is?) /etc/rc.local but that stopped working the first time systemd was on my machine. there's a different file for graphical programs.

installation was simple: **sys a: b:** would copy the system files from the first drive to the second drive. now just copy any other files you want (all of them?) to the new drive-- youre done. "**installation**" **complete**. only took two commands:

sys a: b:

copy a:*.* b:

though you can do this with a single **dd** command if you have a hybrid iso.

returning to the 21st century

does this mean you should have a command-line only distro? not unless you want one-- the point is, that most tasks were incredibly (ridiculously) simple in terms of steps involved:

* **name a program to run it**

* **type three letters to list files**

* **type five letters to make a disk bootable**

* **type six more to copy the files on the disk**

sure, you had to know what the heck to type. but it was the opposite of fancy, and the commands in gnu/linux are much "worse."

it took me years to adjust to bash from dos commands. and all in all, i dont miss dos commands (but i did for the first year or two.)

these days i still use bash, but i also make my own commands.

and **puppy used to be more about us having fun and doing our own thing.**

what is it now, really? (some of us still have fun and do our own thing-- though if youre worried about the future of puppy, so am i.)

it takes so much more to put a gnu/linux distro together than a dos boot disk.

before i started remastering puppy using my own tools, remastering was a level of magic that i just felt unready for-- its too complicated-- sure, i can run someone elses remastering tool, but **i dont get it**. it just does its thing, and i hope for the best.

you gain a lot of confidence if you can remaster an operating system without using someone elses tool to do so.

before it was-- ok, heres this really cool idea, **if we could just get** these remaster program **wizards** to **make us** this really specific tool to do whatever we want.

now it is more like-- ok, i know how remastering works-- im now certain that we could create a remaster program that rivals all others. because the secrets are unfolded before us.

there is one sort of remastering ive never done-- the most irritating part of remastering is the initrd because it follows more archaic/esoteric rules.

there are scripts for that in puppy, im just very stubborn and want to make my own tools (which work the way i want.) but the remastering i do doesnt include changing init.rd, **except** for changes made via debian or trisquel (so far.)

either way, the puppy community has all the code it needs for this. since ive worked on things according to my needs and interest, ive managed to avoid directly hacking the initrd and when ive needed to change it ive done so through 3rd party programs. no problem.

here is remastering in a nutshell: (one way to do it)

1. get the iso
2. mount the iso
3. **unsquashfs** the sfs you want to change
4. **change the files and mksquashfs** to recreate the sfs
5. use xorriso or **genisoimage** or mkisofs to create the new iso
6. run the **isohybrid** program from syslinux to make the iso dd-able to a usb

you can create a program that does these 6 things, and you can have step 4 do all sorts of wonderful things. but it mostly comes down to copying files.

where it gets complicated

when barry started doing this, having fun and doing his own thing was a large part of it.

yes, he looked for friendly-to-use and lightweight tools over others-- so should you. he looked for options to make more fit on a small image-- thats a standard practice in (some parts of) the tech industry. sometimes there is a balance to be found between tiny and friendly, or tiny and fast (because of compression speed vs compression size.)

then he created enough code to make it work together, and it didnt more complicated until feature, after feature, after feature was added.

there are disciplined ways to do that, and undisciplined ways. what happened with puppy is that everything became tightly integrated--

puppy is not designed to have parts swapped out and replaced without major surgery. the stuff that petget does isnt rocket science-- its just very specific. but for example, if you want to replace the dialog-- you can do it, youre going to have to get deep enough into the code. and at least a lot of it is interpreted--

i mean it could have all been done in c, but barry likes doing puppy development with bash scripts. none of those need to be recompiled (some of the tools do, naturally. but not most of the scripts that make puppy puppy.)

it is those bash scripts that puppy developers are struggling with now-- not because they dont know what theyre doing, but because the bash scripts were never made to be decoupled or have typical people maintain them. and just to be clear, if i had written puppy it would have all the same problems in this regard. but **bash is a very unforgiving language to do this in.**

let me show you what i mean-- and **dont worry if you dont understand the code-- lots of people write better bash code than i do**, and thats sort of my point. but i spent many years learning bash, at a time when i actually needed to, and weve had our differences. it does some really cool stuff, but it is designed to be flexible, not friendly. people learn it because they need it-- but they learn a lot of unreliable (and tedious) ways of doing things.

here is one of my favourite bash programs-- it is an implementation of a simple dialect of logo, and a program written in that dialect.

the actual program that the bash program inteprets is:

```
pu u7 l24 c7 pd r 46 d 19 l 46 u 19 pu home

c2 pu l 20 u7 pd d8 r4 ur1 u2 lu1l4 pu

c1 r9 pd r3 rd1 d2 rd1 lu1 ld1 l2 lu 1 u2 ur1 pu

c3 r 8 pd r3 l2 ld 1 rd1 r 1 rd 1 ld 1 l2 pu

c5 r 8 pd l1 u 7 d3 r4 rd1 d3 pu

c4 l 16 d1 pd d6 pu

c6 r 4 pd r3 pd ur 1 u1 lu 1 l2 ld1 d1 rd 1 pu

c4 r 7 pd r3 pd ur 1 u1 lu 1 l2 ld1 d1 rd 1 r3 d2 ld 1 l2
pu     u2 l3 r2 ur 1

c6 r 7 pd r3 pd ur 1 u1 lu 1 l2 ld1 d1 rd 1 pu
```

how to read that program:

pu means "pen up" (dont draw, just move.) **pd** means "pen down." (draw.)

c means change **color**, **0** to **7** are **black, blue, green, cyan, red, magenta, brown, white**.

u means move **up**, **d** means move **down**, **l** means move **left**, **r** means move **right**.

you can go diagonally just by saying two letters first: **lu** means "go diagonally **left and up**."

the number after the direction is how far to move. **ld1** means go left and down **1**.

here is the bash program that interprets that dialect-- just to make a point:

```
#!/bin/bash

proginf="bashlogo, jun 2017 mn"

x=40 ; y=10 ; pendown=1 ; pencolour=7 ; ud=0 ; lr=0 ;
numeric=0

function ne() {

    nl="$nl" }
```

```
function xy() {
    xy_x=${1} ; xy_y=${2}
    echo -ne "\\033[$xy_y;$xy_x""H"
    }


function right() { # probably a better way to do this, but this one was a lot of fun
    right_buf="" ; right_c=0
    right_p=${1} ; right_x=${2}
    for right_b in $(echo "$right_p" | tr ' ' '_' | rev | fold -sw 1)
    do right_buf="$right_buf$right_b" ; right_c="$((($right_c+1))" ; if [[ ! "$right_c" -lt "$right_x" ]]
    then break ; fi ; done ; echo "$right_buf" | rev | tr '_' ' '
    }


function colour() {
    colour_f=${1} ; colour_b=${2}
    if [[ "$colour_f" == "" ]] ; then colour_f="0" ; fi ; if [[ "$colour_b" == "" ]] ; then colour_b="0" ; fi ; colour_n="0"
    if   [[ "$colour_f" -gt "7" ]] ; then colour_n="1" ; colour_f="$(($colour_f-8))" ; fi
    if   [[ "$colour_f" == "1" ]] ;  then colour_f="4" ## switch ansi colours for qb colours
    elif [[ "$colour_f" == "4" ]] ;  then colour_f="1" ; fi ## 1 = blue not red, 4 = red not blue, etc.
    if   [[ "$colour_f" == "3" ]] ;  then colour_f="6"
    elif [[ "$colour_f" == "6" ]] ;  then colour_f="3" ; fi
```

```
    if   [[ "$colour_b" -gt "7" ]] ; then colour_b="$
(($colour_b-8))" ; fi
    if   [[ "$colour_b" == "1" ]] ;  then colour_b="4"
    elif [[ "$colour_b" == "4" ]] ;  then colour_b="1" ; fi
    if   [[ "$colour_b" == "3" ]] ;  then colour_b="6"
    elif [[ "$colour_b" == "6" ]] ;  then colour_b="3" ; fi
    echo -ne "\\033[$colour_n;$((30+$colour_f));$((40+
$colour_b))m"
    }


function lmv() {
    lmv_ud=${1} ; lmv_lr=${2} ; lmv_num=${3}
    if [[ ! "$ud" == "0" || ! "$lr" == "0" ]] ; then
    for lmv_r in $(seq 1 $lmv_num)
    do x=$(($x+$lr)) ; y=$(($y+$ud))
    if [[ "$pendown" == "1" ]]
    then xy "$x" "$y" ; colour 0 "$pencolour" ; echo -n " "
; colour 7 0 #echo "$lmv_ud" "$lmv_lr" "$lmv_num" "$y" "$x"
    fi
    done
    fi
    }


clear ; colour 11 0 ; echo $proginf ; colour 7 0 ; xy "$x"
"$y"


program="pu u7 l24 c7 pd r 46 d 19 l 46 u 19 pu home "
program="$program c2 pu l 20 u7 pd d8 r4 ur1 u2 lu1l4 pu"
```

```
program="$program c1 r9 pd r3 rd1 d2 rd1 lu1 ld1 l2 lu 1 u2 ur1 pu"

program="$program c3 r 8 pd r3 l2 ld 1 rd1 r 1 rd 1 ld 1 l2 pu"

program="$program c5 r 8 pd l1 u 7 d3 r4 rd1 d3 pu "

program="$program c4 l 16 d1 pd d6 pu"

program="$program c6 r 4 pd r3 pd ur 1 u1 lu 1 l2 ld1 d1 rd 1 pu"

program="$program c4 r 7 pd r3 pd ur 1 u1 lu 1 l2 ld1 d1 rd 1 r3 d2 ld 1 l2 pu     u2 l3 r2 ur 1"

program="$program c6 r 7 pd r3 pd ur 1 u1 lu 1 l2 ld1 d1 rd 1 pu"

buf=""

for prs in $(echo "$program" | tr A-Z a-z | tr '\n' ' ' | tr ' ' '_' | fold -sw 1)

do buf="$buf$prs" #; echo $buf | tr '_' ' ' ;  r7="$(right ${buf} 7)" ; r6="$(right ${buf} 6)" ; r5="$(right ${buf} 5)"

r4="$(right ${buf} 4)" ; r3="$(right ${buf} 3)" ; r2="$(right ${buf} 2)" ; r1="$(right ${buf} 1)"


if [[ "$r4" == "home" ]] ; then x=40 ; y=10 ; buf="" ; ne home ; fi

if [[ "$r2" == "pu" && "$pendown" == "1" ]] ; then pendown=0 ; buf="" ; ne pu ; fi

if [[ "$r2" == "pd" && "$pendown" == "0" ]] ; then pendown=1 ; buf="" ; ne pd ; fi


if [[ "$r2" == "c0" ]] ; then pencolour=0 ; colour 7 0 ; buf="" ; ne black ; fi

if [[ "$r2" == "c1" ]] ; then pencolour=1 ; colour 7 1 ; buf="" ; ne blue ; fi
```

```
if [[ "$r2" == "c2" ]] ; then pencolour=2 ; colour 7 2 ;
buf="" ; ne green ; fi

if [[ "$r2" == "c3" ]] ; then pencolour=3 ; colour 7 3 ;
buf="" ; ne cyan ; fi

if [[ "$r2" == "c4" ]] ; then pencolour=4 ; colour 7 4 ;
buf="" ; ne red ; fi

if [[ "$r2" == "c5" ]] ; then pencolour=5 ; colour 7 5 ;
buf="" ; ne magenta ; fi

if [[ "$r2" == "c6" ]] ; then pencolour=6 ; colour 7 6 ;
buf="" ; ne brown ; fi

if [[ "$r2" == "c7" ]] ; then pencolour=7 ; colour 1 7 ;
buf="" ; ne white  ; fi


if [[ "$buf" == "" ]] ; then b=""

else


if [[ "$r1" == "0" || "$r1" == "1" || "$r1" == "2" || "$r1"
== "3" || "$r1" == "4" || "$r1" == "5" || "$r1" == "6" ||
"$r1" == "7" || "$r1" == "8" || "$r1" == "9" ]]

    then if [[ "$numeric" == "0" ]] ; then num="$r1" ;
numeric=1 ; else num="$num$r1" ; fi

    else if [[ "$numeric" == "1" ]] ; then lmv "$ud" "$lr"
"$num" ; numeric=0 ; ud=0 ; lr=0 ; fi

    fi

if [[ "$r1" == "u" && ! "$r2" == "pu" ]] ; then ud=-1 ; fi

if [[ "$r1" == "d" && ! "$r2" == "pd" ]] ; then ud=1 ; fi

if [[ "$r1" == "l" ]] ; then lr=-1 ; fi

if [[ "$r1" == "r" ]] ; then lr=1 ; fi

fi

done ; colour 7 0 ; echo
```

this wasnt written to be the loveliest bash script ever-- it was written to perform a fun and silly task as part of a contest to implement a programming language in 100 (physical) lines of code. (there were other limits-- one obscenely long line of c code wasnt going to do.)

this was written just for fun, and i bet someone could knock this monstrosity down to a couple of lines:

```
function right() { # probably a better way to do this, but this one was a lot of fun
    right_buf="" ; right_c=0
    right_p=${1} ; right_x=${2}
    for right_b in $(echo "$right_p" | tr ' ' '_' | rev | fold -sw 1)
    do right_buf="$right_buf$right_b" ; right_c="$((($right_c+1))" ; if [[ ! "$right_c" -lt "$right_x" ]]
    then break ; fi ; done ; echo "$right_buf" | rev | tr '_' ' '
    }
```

but i was happy once it worked. if youve worked to really modify complex scripts for puppy before, youve probably encountered something as ugly as this. bash is not ideal for very long, complicated scripts-- it can do a lot, but it cant do everything well. for some tasks that are trivial in other languages, bash can be very demanding and high mantenance.

high maintenance counts when youre trying to organise all the code in puppy.

lets take these 4 lines:

```
    do x=$(($x+$lr)) ; y=$(($y+$ud))
    if [[ "$pendown" == "1" ]]
    then xy "$x" "$y" ; colour 0 "$pencolour" ; echo -n " " ; colour 7 0
    fi
```

this code could easily look more like:

```
    x += lr
    y += ud
    ifequal pendown "1"
    p xy x y ; colour 0 pencolour ; " " prints ; colour 7 0
    next
```

something as straightforward as "increase x by lr" this in some languages:

```
x += lr
```

becomes this in bash:

```
x=$(($x+$lr))
```

bash has other factors that make it less friendly-- and again, this isnt to knock what bash is good at-- ultimately you get a rich, flexible system that ties together command line tools, though there are several ways to break the code that make it more tedious to use.

for example this line, in more than one language:

```
x += lr
```

works whether you write it any of the following ways:

```
x += lr
x = x + lr
x+= lr
x +=lr
x= x+lr
```

while bash can be unusually sensitive about whitespace:

```
x=$(($x+$lr)) # works
x= $(($x+$lr)) # command not found
```

the reason of course is that bash is a langauge for creating command lines-- a language to use as the shell. evaluating expressions is a secondary feature, processing entire strings is not as high a priority as dealing with tokens.

but this makes it a more complicated language to deal with-- there are countless examples of gotchas and special tricks to learn if you want bash to do exactly what you want. a friendlier language would not have so many of these.

more than a decade into learning bash, i still marvel at those who have greater skill in bash than i do-- i can find many of these tricks in scripts, but i dont know what all of them are for nor do i know the easiest way to solve every task.

another pain of dealing with bash is that **the commands are not standard**. bash is a standard, but busybox is different, and you dont know which commands will be available on a given machine, nor whether the version of that command is the one your code works with, and so on. finally, function scope in bash is a pain for large scripts, because you have to keep track of too many variable names:

```
function right() {
    right_p=${1} ; right_x=${2}
    # this is how you get the first and second parameters
    # sent to the right() function in bash
    }
function right(right_p, right_x) {
    // this is how you get the first and second parameters
    // sent to the right() function in javascript
    }
function right_ right_p right_x
    # this is how you get the first and second parameters
    # sent to the right_() function in fig
    fig
def python_right(right_p, right_x):
    # this is how python would send parameters to right()
```

the differences are not just in syntax, but in scope. for javascript, fig and python (as well as basic) functions have local scope.

what this means in practical terms:

```
function right_ right_p right_x
    # the values of right_p and right_x for the rest of
    # the program do not matter-- they are separate here.
    # only the value of right_ will change.
    # any variable names used in this function will not
    # affect values outside the function.
    fig
```

with bash, any changes to variables **inside** a function will affect the rest of the script.

this means that you have to either keep track of every variable in the entire script-- the larger the script gets, the more difficult this is-- or you have to use a really ugly and tedious workaround like prepending each variable name with a pretend namespace:

instead of **right_p** and **right_x** for names, you can simulate local variables with worse names like **fig_right_right_p** and **fig_right_right_x**.

after the 1960s and 70s, it became more common for languages to provide local scope for functions, which saves you loads of trouble. and for smaller scripts this doesnt matter-- you can still have local scope if you split large scripts into small ones.

but all these factors combine into a harder environment for many to develop in.technically these are things you need to know **to maintain puppy**-- but that isnt a given, and it doesnt have to be that way. we can actually do quite a lot to work around all this, and make it so that more people can maintain puppy.

expect this to cause debate about which points are entirely true (about bash) and which can be mitigated. i pointed out one way to mitigate the part about local scope, but these debates about bash can be endless and are also tedious. the real point here is: **we can make it much easier to maintain puppy.**

but what about legacy puppy?

ok, so this is where we decide what to do about all the legacy features in puppy.

and the answer is: **whatever you want**-- the concepts outlined here are pretty much agnostic to new puppy vs. old puppy. ideally, we would kickstart a new era of puppy development that actually covers **both** old and new stuff.

sure there are new approaches outlined here-- there are a lot of those, but these are new approaches that take into account the idea that **everybody wants something different.**

indeed there are at least two things about the reality of puppy that are universally true:

1. puppy is not a professionally developed distro.

puppy is a hobbyist distro. thats why people were having fun. people arent paid to develop, most of the developers dont come from professional developer backgrounds, the ethos behind puppy from day one through the day that barry "retired" is one of hobbist development. to kid ourselves about this is to misunderstand puppy.

2. that doesnt mean we cant have nice things.

free software means you dont have to do everything yourself. you can take professional-quality (at least, more professional-quality) software and get puppy to make use of it, to include it, to (some extent) integrate it. where more professional quality software is too slow, too large, not fun or friendly enough, we can make our own alternatives-- its mix and match, puppy did this almost from the beginning.

and **how we put all this stuff together** is largely what its about. from what im told, the woof devs work very hard to try to make people happy, though they are increasingly on about **their own way** of doing it.

its difficult to explain to people (for them, for anybody) how **doing things differently** can help you **do more things your own way**.

in fact that may not be the goal of the devs anymore at all-- barry was pretty much going to do things his way, he had the final say so to speak-- the community gave him suggestions until he was tired but ultimately he found a new way to do things his way.

its difficult for any developers to cater to the myriad wants and needs of every possible user. in the past this was done by keeping puppy modest. the more people like it, the more they brag that it can do anything, the more defensive the community got about limitations-- the harder it is to fix things.

if this sounds like a defense of the developers-- it sort of is-- but only part way. because the developers have limitations, and part of that is because **everybody wants something**-- but also, at some point somebody wants to work past the limitations of the developers.

at a certain point, developers become prone to ignoring or disputing more and more of the wants and needs of users.

at a certain point, users become prone to ignoring or disputing the reality of the developers.

the users dont know what to do about this, and if i had to guess, the developers dont know what to do about this either.

so what i will offer you is a technically-informed philosophy (inspired by puppy) that is designed to give everybody everything. like all things in computing, it will have actual limitations and fall short of perfection. but we will start by aiming for perfection, which is a great place to start sometimes.

woof is not the reality of puppy development, woof is the official aspect of puppy development. everything unofficial is the unofficial aspect of puppy development. since we cant (and certainly wont) make the official developers adopt any of our philosophy, this will need to focus on the unofficial builders. in fact, this entire writing assumes that woof devs will never abandon github, and that microsoft will ultimately abuse users to the point where the community runs from github just as they used puppy to get away from windows.

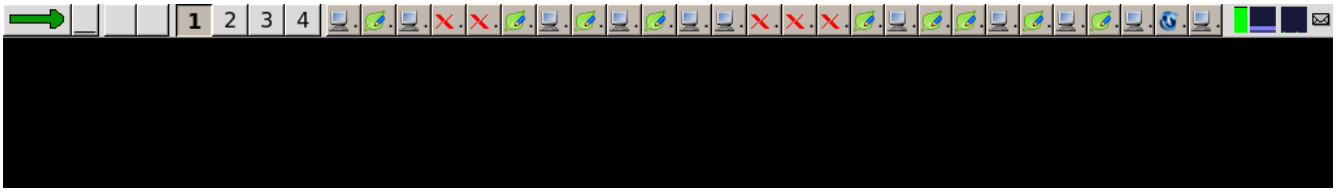
the title after all, is **recommendations for salvaging puppy linux**. what are we salvaging it from? github, and a future where people arent capable of maintaining puppy or taking it to new places.

unofficially, people have "cross-bred" puppy to create all sorts of wonderful mutts. i worked on mixing puppy with refracta, librepup mixed puppy with devuan, debiandog is based on debian of course, slacko is official but based on slackware. one of the things i worked on was getting pet package support into refracta (devuan.) that was fun, and something only a hobbyist (never a professional developer working for money) would probably ever do.

so officially, puppy is going to go in a **single** direction and unofficially, puppy is going to go in **several** directions..

the puppy community has always catered to that and probably always will. will the recommendations here render the work of the woof devs useless? no-- but it will not depend on them either. some of the ideas may help them quite a lot, but the goal isnt to help or hurt them-- the goal is to be free.

as for legacy, **i use a window manager from 1998**.



i talk about the simplicity of a dos boot disk (ive got a lot more to say about that) as an ideal model of simplicity for modern distributions. but that doesnt mean im going to try to come up with solutions for puppy that focus entirely on legacy, because people are still going to want to use things they consider nicer.

but:

2. **if you want to promote free software, collect computers-- you dont have to save them all, you dont have to keep more than a few on hand, collecting them will give you more opportunities to get inexpensive hardware when you need it; you can help people find cheap hardware when they need it.** https://freemedia.neocities.org/zero-dollar-laptop.html

3. if you have more than one computer**, have a lightweight software platform for at least one of them-- it will be most useful for older computers, make work faster on others**, and make it more obvious what to do with older machines. https://freemedia.neocities.org/distro-libre-and-feature-schema.html

from **https://freemedia.neocities.org/best-practices.html**

we want to be able to salvage (and update, and preserve) **old versions of puppy** when it suits us.

that point is largely a matter of remastering, and remixing.

so all of this continues to be about how to remaster, remix, and build puppy.

it is also about **documentation** and **education**.

puppy schools and documentation

one of the best ways to promote software is to teach, and one of the best ways to design software is to **document it first**.

if you want to contribute to the wiki, puppy has a wiki you can contribute to. if for some reason you dont want to do that, here is another way:

go to **https://neocities.org/** and create an account. you can host **html, images and javascript**. if you would rather host pdfs and odt, a better place to register is the internet archive: **https://archive.org/account/signup**

if you choose neocities, i have some javascript you can use to make a template for each page-- **this template lets you edit each page like a wiki** instead of bothering with html. unlike most wikis, **this one lets you make changes to the syntax** (because it is implemented in javascript.) so you can actually have your own custom wiki that works the way you prefer.

i stopped enjoying html edits years ago, i prefer wiki syntax.

if you would rather use openoffice or libreoffice to create pdfs, archive.org has a cool way to host those (but you should upload the .odt file as well.)

now, why host your own documentation?

first of all, hosting your own documentation lets you be specific about what version(s) of puppy you are supporting. i would recommend if you write any documentation (even a quick how-to) for using your favourite programs, that you choose to document things that are **stable** and hvae worked the same for a while. this encourages stability and consistency in development.

one of the worst things about using windows was getting dragged through fad after fad, designed to force people to upgrade windows to the latest suites, even when they dont need to. security updates are another matter-- my window manager from the late 90s is almost exactly the same as it was then, but it has received minor updates and maintenance over the years.

professional development requires feature churn, which gradually takes things away from the user. this is more the fault of people outside puppy development than it is the fault of puppy developers-- as the distros we depend on change things, puppy has to choose between staying with older versions (which it often does) and eventually, making things work with newer versions. but we can

resist this to some degree- for example, devuan resists systemd and helps sustain the possibility of a future where systemd is never required. thankfully, puppy does this as well.

by documenting and promoting **stable** applications, you can help make puppy relatively stable. this is not to say that there is no place for newer applications-- rather an attitude that "newer isnt always better" is part of what makes puppy puppy, and while integrating newer applications is sometimes also a thing with puppy, minding the tried-and-true is something we probably dont want to abandon.

another way to say this is that keeping an old, reliable window manager like jwm working is a **higher priority** for many of us than getting the latest version of kde to work. getting kde to work is going to be a higher priority for some people, and thats fine too.

we want puppy to keep a stable foundation that we can build things from. we have no idea what people will build using puppy in the future, and thats one of the exciting things about it. but for us, its nice if we dont have to keep learning puppy over and over and over again.

when things do change, they should get easier-- and they should get easier in a way that doesnt require kneeling before wizards. because even if we have generous, friendly wizards, eventually they get old and then we have to scramble to find new ones-- good wizards are hard to find, and they dont realy like being controlled by the community. so we need to find a way to create more apprentices, and lean on wizards a little bit less.

another way to say this is that any magic we create should be easy to share.

the way that puppy was in the earlier days, is that barry was the wizard and there were more apprentices, who were an integral part of the community. the wizard worked to impress everyone, the apprentices helped and contributed, and everybody else celebrated puppy and asked for things.

that arrangement still exists, but shows plenty of signs of wear. the premise of this entire writing is that there arent enough apprentices, and to have more we should make it (much, much) easier to become one. we make that job title easier by making the process easier. we make the process easier by making puppy easier to maintain.

so one of the things we want to document is how to use applications we love, and another thing we want to document is how to change (maintain) puppy.

the other nice thing about documentation, is that you can have a section for documenting how things **should** work. rather than how-to documents, these are design documents, and if you write a good design document it could lead to better tools.

so even if you cant code (and if youre under 50-- maybe even if youre over 50-- you should try to learn something at least) you can still write down your ideas about how puppy could work.

a discussion forum is a natural place to do that, but then it turns into a debate. debate is good, but sometimes it prevents ideas from being duly considered. if more people documented how software **should** work, i strongly believe it would ultimately result in better software.

even better, the documentation you write could be put under the same licence as wikipedia (**cc by-sa**) or put under a public domain waiver: **creative commons cc0 1.0 (public domain)** which is compatible with cc by-sa, and then other people can maintain and reuse (and reorganise) your documentation in the future.

doing things this way could result in more collaboration and a community more prepared to help maintain puppy.

and although it isnt discussed in much detail yet, the best part is not having to spend many years learning the countless ins and outs of bash.

even if you dont code, you can document puppy. even if you arent an expert, you can write easy how-tos for people who also arent experts.

even if you dont want to work on the wiki, you can do your own thing and put it under a license that makes it so your work helps anybody who can use the documentation you put together-- just like wikipedia does. only without the edit wars and tiring talk pages.

and this isnt to make the forum useless, either. there will still be plenty to talk about on the forums-- there always was.

this is about creating one more option.

when you design software, **dont worry if your idea is feasible or not**. when star trek invented the ipad, the necessary hardware didnt exist for decades. when i thought it would be cool to use chips instead of floppies, eeproms were far from a practical way for users to store files. if you cant implement, theorise. but by all means, learn more about the things you cant do-- because it will make

your theories and design ideas better-- even if you decide to just treat coding as some science-fiction concept. thats alright, science fiction gave us the cell phone. lobster used to write about a version of puppy that travelled through time: https://archive.org/details/Puppy_Linux_tmxxine

here is some generic information about artificial intelligence, present and future, including possible effects on free software and gnu/linux: http://techrights.org/2019/08/20/enhanced-damage/

here is alan kays laptop concept from 1972: https://en.wikipedia.org/wiki/Dynabook while we have had laptops for some time, dynabook was also an **educational** concept. when the one laptop per child project started decades later, kay joined and helped make it the closest thing ever to the dynabook concept.

by all means, consider the practical and the present-- but dont limit yourself to those thoughts alone. one of the most enjoyable things you can do with a computer is imagine the future. find a way to talk about this online, and if you can do that on the puppy forum-- good. if that doesnt work, **find a place** where you can. this "puppy of the future" at best, is not disconnected from the past, but also **affirms** the legacy and spirit of puppy. but how it does that is up to you. ill provide some further ideas about it.

puppy anatomy

people who take puppy apart more than i do can comment on this with more accuracy and detail. i will make generalisations that give you a basic idea, based on taking apart several old and several new standard versions. you can document the anatomy of your favourite version(s) of puppy on whatever level is important to you: the visual level, the component level, or the one that you need to remix, remaster or build puppy.

1. **isolinux**

isolinux is part of syslinux, and while you can use grub and its config file(s) to boot puppy, isolinux is the thing that boots puppy from the iso. **isohybrid** (also from syslinux) is a program that will modify your iso so that it can be installed to a usb using only dd. there are several bootloaders that work with puppy, but if you remaster the live iso then isolinux.cfg is a file you might edit.

2. **initrd**

the most complicated part of puppy to modify other than the kernel, the initrd is the image that gets puppy started. the kernel is in one file, initrd is another file, and initrd contains what is needed to do things like load the sfs files. if you mind the sizes, you can move everything from the sfs into the initrd-- that isnt typical, but some lightweight distros (and pups, i think) have tried it.

3. **sfs files**

most of the things you want to remaster are in the **sfs** files. they are fairly easy to open and change and resquash. like a tar.gz, sfs is a compressed filesystem. the difference is that with sfs, you have the choice to mount the file as a filesystem instead of decompressing it first.

these are conventions-- you'll find them in puppy, debian, devuan and refracta.

4. **xvesa, xorg** and **wayland**

puppy used to give you a choice between **xvesa** and **xorg**, these days **xorg** is pretty standard and **wayland** is trying to take over. these provide the graphical layer for puppy, but not the graphical shell or desktop. can you have puppy without this layer? absolutely, though versions of puppy without the graphical layer are rare by comparison.

5. **graphical login / dm**

puppy typically goes right to the desktop. this is an option of several display managers, though most of them are used to let you select a window manager or desktop and to login. so this is probably a feature of some versions of puppy, though probably not one you encounter in daily use, except for unofficial versions.

6. **window manager or desktop environment** (read: jwm and rox-filer)

i fell out of love with jwm and rox years ago, but they are still good demonstrations of what i want in a desktop-- light, fast, simple. thats hard to do in a way that everyone likes-- which is why there are other desktops as well. but i still love icewm the best-- here is a clue to the specs of what im typing this on:

```
# free -m | cut -b 1-48 # no swap
              total        used        free
Mem:           7788        1027        4972
```

so do i "need" a window manager from 1998 that uses **zero-point-one percent** of the ram? no, but i dont like wasting resources on the ui and icewm has the features i want and need. why would i slow the thing down with something heavier? ive got openoffice AND libreoffice open (no, you dont need both) and most of the 1gb of used ram is because of the web browser.

granted some people want a big fancy desktop. and after we have a minimal thing that works on old machines too, theres no reason not to have packages or distros that accomodate that. though personally im pretty tired of both gnome and kde. i like the old lxde and i like xfce-- their development values (so far) are closer to my own. i really want to like lxqt but so far the only thing i like is the qt version of pcmanf, which is about the same as the gtk version.

7. **widgets and wizards and packaging**

puppy has petget, and i remember pupget. i guess it was confusing when distros were pups and packages were dotpups. puppy packages are easy to take apart, and not impossible to put together. theyre easier to manage than .deb packages, though you can unpack .deb packages with dpkg-deb which is included in some official puppy versions. this is great if you want to examine a .deb without installing it first, or if you want to install it without a full apt setup.

speaking generally, packages are basically:

1. files to install

2. metadata / lists of files to install / uninstall

3. scripts to run (optional)

4. tools to create, install and manage packages

puppy has a lot of other friendly utilities, some of which are integrated with jwm or tightly integrated with rox-filer, which provides the desktop icons for puppy.

this is not always true for unofficial versions of puppy-- i like pcmanfm over rox, icewm over jwm, and there are other people who prefer these setups as well-- but changing the desktop doesnt always give you full access to the complete puppy experience. some of us are ok with that, others would not be. this is an area where we can certainly cater to both groups.

8. **remaster tools and build tools**

these deserve their own category, and arent always included with puppy. not every version of puppy includes woof. originally, puppy remastered itself with

various methods of appending to the live-cd or usb. there are many ways to do this-- some brilliant, some unsophisticated (simple.) for example, my own distro mounts certain partitions as read-only, and if it finds /etc/.figos-local on one of those partitions, it runs /etc/rc.local from the hard drive-- that can then mount only the partitions i want it to, and even replace the home folder via a symlink.

thats a very **unsophisticated** way to have a distro that "remasters" itself (this isnt remastering technically, it is persistent settings via a partition. but it does so with two fairly simple files-- /etc/rc.local on the distro, and a second one on the partition.)

puppy has included several tools for several ways of doing this. some are pretty complex or sophisticated, and have spent years being tweaked to what was once considered perfection.

remastering takes an iso (when i do it) or sometimes the current installation (this is also known as a snapshot) then creates a new iso. a build tool can create a new iso without the current installation (it certainly needs some kind of installation, but will build based something else) and in some instances, without pre-compiled binaries.

if we really want puppy to belong to everyone, to suit the entire community, it is this category of tools that the community will need to promote and/or create, and eventually use.

9. **other applications**

the stuff that makes it a gnu/linux distro-- text editors, programming environments, solitaire games, a web browser-- you know, **stuff** that people use. if we really want puppy to belong to everyone, it is a very good idea if everyone documents what applications they like-- at least. as well as other things. the section on "**puppy schools and documentation**" covers this, but for those who care about more than just applications, it would be wise to document the aspects of puppy (of any kind) that mean the most to them.

attitudes in the community

young trolls and old curmudgeons are largely different breeds, with different levels of patience for different annoyances.

the puppy community has mellowed as both its users and puppy itself have aged. i first used puppy more than a decade ago, and dual booted it with windows 9x and grub4dos.

if you think the community is rough now, i think the flamewars used to be a lot worse.

puppy has lost a number of sensitive and talented people (you could almost argue that barry is one of them, though he is not nearly as sensitive as some of us) to bullying and trolling. if youre expecting a code of conduct to come out of this, i dont believe in them:

http://techrights.org/wiki/index.php/Librethreat_Database#Code_of_Conduct

- Summary: **Can be abused to stifle and silence important feedback**
- Mitigation: **Adopt more reasonable version, avoid altogether, address same problems that CoC aims to, but with more allowance for free speech and diversity of opinion**

in the hands of really diabolical trolls, codes of conduct can be used as one more weapon at their disposal. all they have to do is find a very personalised way to harass you that flies under the radar, and then rely on the code of conduct when you finally react to their campaign of needling.

there is only so much you can do about people like that, because a community of diverse ideas that celebrates and desires diverse ideas is really hard to build (and harder to maintain.) ultimately some people will never be happy no matter what, and youll have to figure out what to do with them. if they are worth the trouble, they will probably be tolerated.

linus torvalds for example, is not in my opinion a very nice person. i consider him dishonest and unfair **about certain things** and about certain people. but i also think he when it comes to developing his own kernel, he has more integrity than other developers-- we wont be better off when he leaves. i not only tolerate him, i will defend him-- even though i really, really dont like him. i like his kernel.

there are worse people than linus, who contribute nothing-- like the people he works for, that dont even use gnu/linux (they use macs. torvalds boss is a mac user. so is the guy that maintains linux.com-- and theyre both huge fans of microsoft as well!)

so when youre talking about bullies (and bully isnt what i would label linus as anyway-- its not the way he talks to developers that bothers me) do you go after the smaller bullies or the larger ones? because a lot of times, people will focus on the little bullies they can isolate, and ignore the ones at the top who consolidate power picking on the smaller ones with everyone else.

when trying your best to avoid drama doesnt work, sometimes it finds you-- and you can try to stand up for yourself, you can escape and go somewhere else, you can ask for someone to wield their club on your behalf, though mostly communities are imperfect and if you have enough controversial ideas, you will be looking to strike that perfect balance between **not caring what people think** and standing up for yourself when you have no other choice.

the best way to not care is to be independent of the community-- interact, but dont rely on it. and how do you create a community that provides, but is not relied on? one of the best ways to do that is to **decentralise** the community-- make it so that it is controlled by a larger number of good people in more than one venue, not by one good person charged with being a benevolent / hands-off / personal / on-call dictator. its honestly a wonder that ours has survived with his sanity intact, its a nearly impossible job (try it.) even ive put him through a lot.

but with a decentralised community or federation, you get multiple chances to gain acceptance, a voice and allies. you also reduce the strain on the community leaders. and instead of the community circling the wagons whenever an idea is introduced, there is more than one venue for it to inspire people.

plus, if one subcommunity isnt so nice at the moment, you can take time off without quitting.

it probably helps if the subcommunities arent completely isolated-- if there are still ways to have some level of communcation (and advocacy) between them.

the other way to be independent of the community is for things to be designed less for developers only-- but this requires a more educated userbase and sometimes, better tool designs.

d-i-y-t: d-i-y together

when you have a problem, you shouldnt have to throw yourself on the mercy of an entire community. thats typical, but its also overkill and works better in theory than practice.

problems should be easier to fix, and require fewer experts. this may sound like one of those "easier said than done" things, but it isnt that as much as it sounds like-- problems sometimes get harder to fix, and this is the real problem.

puppy (like most things) has grown increasingly complex over the years, and the more complex it gets, the more of an expert you have to be to fix things. if you rely exclusively on abstractions and community to manage this complexity, youre going to have an impatient, overly taxed community and you will be entirely at the mercy of the people who maintain the abstractions.

consider every feature of puppy as overhead-- something that costs effort to maintain. if you have a small number of official developers, you can either charge them with maintaining everything, charge them with simplifying, or you can get involved.

traditionally getting involved means coding, and ive already outlined alternatives to that. in puppy, coding typically also means writing bash scripts, and i will outline alternatives to that also. (i talk about alternatives to that all the time, so if you want to know how, all you need to do is take interest. but i will also talk about it here.)

if you put a small number of people in charge of an infinite number of tasks, sooner or later theyre going to figure out how to ignore you, cut corners, or both. you can actually help keep puppy simple by figuring out how it could be simpler, but to do so youre going to have to organise a subcommunity for that purpose-- if you do that within the main community, youre going to start a coup-- and with it the effort (not by barry or moderators, only by the community itself) to shut it down.

but just as much as you can help figure out what to cut back, you can figure out what to keep:

1. talk about what you want

2. talk about what you dont need

3. talk about how to reengineer puppy

4. if possible, figure out a way you can help

5. figure out how to build communities that can tolerate your ideas

if you really want to do these things, it has to be something you believe in enough to talk about or work on without any guarantee that it will change things.

in other words, the people most likely to really change something are the ones that are willing to do that **just for themselves.** to make puppy into something more people can change themselves, it has to be simplified again. abstractions can only do so much-- you have to be willing to say "this is too much" and replace something with a simpler version.

documenting what you really like about it can help you design a version that is simpler but does what you need.

the thing that would help preserve puppy the most would be if the parts were less integrated and more modular, though its precisely some of those integrations that make it so cool.

so what can be done about that? find out what everybody cares about the most, find ways for the exceptions to be catered to by someone-- create simpler ways for people to integrate things themselves.

this is as much about preserving functionality as simplifying it, but that cant be done by a smaller group of developers-- there needs to be more people working on the problem one way or another, even if theyre working on their own.

the old adage "help people help themselves" describes what the puppy community did for so many people-- it helped them learn gnu/linux with a distro that was easier to use than most distros. puppy created "experts" on how to use puppy-- and puppy is made of simplified gnu/linux features-- pupget and petget dont do everything apt does, for example.

so if you really wanted to infuse modern puppy with the original puppy spirit, you would be looking for ways to simplify puppy and make it easier to become expert with it. but people will never be satisifed with the outcome of the democratic process, so subcommunities (which puppy already has) around different versions (which puppy already has) are the real future. help yourself, then help your tribe, then help the broader community and the world. that way it matters a lot less what everyone else thinks.

the boot disk concept

because of libraries, and because of the complexity of modern programs, its isnt exactly possible (yet) to abstract a gnu/linux distro down to the simplicity of a boot disk.

you cant just "make it bootable" and then "copy stuff there."

it is almost that simple! but the devil is the in the details.

if you let someone else mess with the initrd for you, (and typically changing the kernel or init system means also changing the initrd, but for example debian does that automatically) then most remastering is just:

1. its already bootable

2. change the files and zip it back up

universal packages may or may not make this easier still, though snap is controlled by canonical (you basically need their help to participate, which is a dead end) which leaves flatpak and appimage-- the latter is probably the most usteful so far.

but rather than have complex tools (long, sophisticated bash scripts) for changing puppy, we want simple, unsophisticated tools.

the way you get those is by keeping development modular, tools simple, and when its necessary to bundle tools together into a "one stop shop" thats fine, you can package it that way but if you develop it that way, sooner or later its going to be unmaintained and abandoned and protested (for being abandoned.)

in other words, if you want one simple tool to create an iso from the running system, what you really want is:

1. a tool to open / save bootable isos

2. a tool to get those settings and copy them to an opened iso

3. a simple command-line tool that uses the first two tools together

4. a gui for the command-line tool

usually people will put those together into a single tool. this is why nothing gets maintained and people just rewrite tangles of impossible code.

its easier to write those tools separately than together. and not only is it easier to maintain-- the part that the second tool does is very specific (to one task) and if you have the first tool you can make other plugins for it-- **instead of a tool to get those settings and copy them to an opened iso,** maybe you want to open a file manager and let people copy files by hand (sometimes.)

then you can add options to the simple command line tool, and whoever maintains the gui can add options to the gui.

you can do all these things yourself if you really want to, but even if you do, it gives those who want to maintain it in the future a much easier time (whether thats you doing it or someone else.)

it makes these tools easier to reuse in future projects.

but what if you really want to put it all together in a single script?

ok, thats how i designed fig (fig is a programming language in a single python script.) but since i want people to be able to take fig apart, or change features or even remove things, i made a guide to doing that at least:

figminus removes features from fig to make it easier to adapt from python to other languages (at which point it probably wouldn't be the same language, it would be fig-inspired.)

this is a copy of fig 4.6, with highlighted code based on the diffs between each version of figminus. the highlighting isnt to make it easier to read-- but to make it easier to find or edit out features.

```
# 4.7 removes pygame / graphics commands
# 4.8 removes commands that require trig or 3rd-party libraries
# 4.9 removes array handling commands or extraneous comments
# 5.0 removes help feature, demo, forin loop, swap and time commands
# 5.1 removes restriction on variable names, errors for wrong command
position
# 5.2 removes filehandling, for float step, try/except/resume/nextin/system
# 5.3 removes string, hex, oct, sgn, sqr, mod, cruft from predecessor of
fig
# 5.4 removes # support, inline python, for loop, reverse, next
# 5.5 removes case insensitivity, shell, chdir, asc, chr, topwr, randint
# 5.6 removes ifless, pass, ifmore, error for reference to unset variable,
instr
# 5.7 removes user-defined functions, wend, plus
# 5.8 removes lcase, ucase, prints, len, rtrim, ltrim, left, right
```

so fig (4.6) has 1150 lines of python code (1700 if you count blank lines) and suppose youre creating a language that doesnt use graphics. just remove the lines that are red `# 4.7 removes pygame / graphics commands` and youll make it a simpler program. the colour-coded guide to the script divides the code into 13 different groups by feature category.

again, this is all optional-- it would simply make puppy easier to maintain. also, everything fig does is process a single text file and output a single text file. if it had a gui, making that a separate program would be a very good idea.

i once spent a very long time (hours?) removing the ide from the source of qb64. if the ide were a seperate program all i would need to do is delete the file. why even remove the ide? every time i made a change to qb64, i had to recompile it. removing the ide made it compile much faster.

collaborative automated remastering

a few tools that could help with remastering puppy in general would be scripts that did certain tasks, such as install or remove gimp.

so for example, you would have a program open the puppy iso, then run a number of tasks:

1. removegimp
2. addpalemoon
3. changewallpaper /root/puppy0102.png

these are scripts that could either install packages or be used as an alternative to creating a package in the first place, and they could be run as plugins for an automated remaster script.

the cool thing about the automated remaster script would be, that instead of downloading the remastered iso, you could have a standard iso with the remaster script, and then just download a text file that lists the plugins-- the remaster script would then:

1. download each plugin, stopping if it couldnt find one

2. ignore cached plugins (not download over them unless they were deleted)

3. run the automated remaster with the plugins

plugins are easier to maintain than a remaster script, and less trouble to download than a new iso.

but the best part is that you could create plugin lists for people based on the preferences they note online.

more and more people can have custom puppies with less work for everyone-- and people who are less technical helping in small ways instead of not at all.

a language for remastering

it is very possible to create a programming language just for remastering, with simple commands for various tasks. such a language can copy files, change configurations, remove packages, and automate various common remastering tasks.

a similar language could be created for **building** pups.

making it easier to automate tasks, using simpler languages, means making it easier to collaborate to do practically everything.

we just dont have the tools designed for everyday people.

and we need to make it so that everyday people can contribute as well.

i suggest a custom language with simple commands, based on a simple language that is easier to work with than bash.

we can have a language like that, but only if people want it.

then the future of puppy is one that more people can learn (more easily) and use to solve their own problems, together.

people have already attempted some things like this, but the community isnt interested yet. there has to be a certain amount of interest-- a certain amount of imagination, and even a certain amount of agreement to make puppy sustainable. fortunately, it doesnt take as much as people think.

free media alliance "thrive" guidelines

version 0.1, 29 july 2019

in a time of great upheaval for both software and community, these guidelines were written:

to
help
realise
ideal
volunteer
efforts

we recognise the threat of even a well-intended "code of conduct" being used to divide communities:
http://techrights.org/wiki/index.php/Librethreat_Database#Code_of_Conduct

rather than a club wielded to stifle important debate or prevent the full resolve of vital differences between individuals or groups, we encourage communities to support both the diversity of opinion and the diversity of contributors.

wherever these guidelines are misused to threaten community and development, they should be regarded with scrutiny-- whenever these guidelines help create a foundation for purposeful development and progress, they should be considered thoughtfully.

1. integrity and checks and balances are more valuable than false compromise.

2. ignoring your own standards, as well as taking rules too seriously, can compromise the integrity of your community. many communities are already diminished along these lines.

3. the corporate monopolies that promise to help resolve these problems, have a history of fundamental selfishness and interference. giving these corporations too great a say in matters has helped them to destroy communities and stifle their efforts.

4. in practical terms, "working together" means finding enough common ground for collaboration. it does not mean abandoning the principles or values of your own community.

5. in dealing with both critics and allies, it is always more useful to look past the superficial-- towards motivations, true nature and real effects. society encourages the shallow evaluation of goods and services, as well as of people. vital communities must do better in this regard than general society, if they wish to thrive. this is not intended to eliminate speculation, only to temper superficiality.

6. without some greater commitment to the needs and education of users, free software will soon lose too much ground to corporations that falsely pander to them. this is not a call to make everything "user friendly." as a user, you are free to develop on your own terms. there are still areas in which progress could be made regarding development.

7. it is better to have communities divided over politics than to have software development and repos hijacked and repurposed by a single political faction.

8. when communities with valuable contributions become divided over political differences, umbrella communities and organisations are a positive way to invite long-term resolution. haste and superficial resolution are less positive, though "first step" efforts will hopefully count for something.

9. each community should be allowed to explore its own options to further the long-term benefits of its efforts towards software freedom-- subject to informal approval and/or intellectually honest (fair) critique from from other communities.

10. communities should avoid, as much as possible and practical, efforts to lock other users into their software or distributions. the more important and popular (and fundamental) the software is, the more modular and

optional and flexible the software should ideally be. even the distro itself should become more modular and universal-- via thoughtful design conventions, rather than rigid and demanding standards. but when in doubt, refer to points 5 and 9.

it is the hope of the alliance that these ideals, when taken as inspiration rather than bylaws-- will help communities move past the long-lasting damage to communities over the past half decade.

in the event of failure of vital communities-- we hope these ideals will not only help rebuild, but push software freedom towards yet-undiscovered levels of success and progress.

distro-libre and feature-schema (borrowed from guarding and rescuing the fsf titanic)

hundreds of distros exist, many of them with very similar features. we know there is duplication of work, but everyone needs to understand why so many distros exist.

every time a distro does not suit a user's purposes, and it is less work to adapt the distro on one's own than to affect the distro in any other way, a distro is born. ego is a factor too, but rarely mentioned is the educational aspect.

if **more** people created distros, then more people would have experience or interest in maintaining (contributing) to existing distros. the real trick is facilitating that.

stallman has said that we don't need more distros. "we" also don't need more text editors, or "hello world" programs. other people say we don't need more programming languages.

each of these arguments are subjective (who is "we?") and can be refuted by pointing to a single need that no distro caters to. but in recent years, many more (once-reliable) distros are lacking than before. are people really saying they don't need to be fixed?

because they are more likely to be repaired by forking. control over distros and of software by monopolies is increasing, and if the halloween documents mean anything then this is a problem the fsf and osi once acknowledged (hosting the documents on their own servers, though osi has removed them since) though **now that it is a more critical and everyday problem**, they are saying nothing about it.

if we need more freedom, then we need more distros. in fact stallman said "we don't need more distros" before the fsf gained hyperbola, one of the very few (and arguably most dedicated) distros to work to remove the monopolistic tentacles of systemd, which guixsd should also be suitable for, but hyperbola should be a lot more friendly and mainstream.

we would say that trisquel probably does not need more distros, but also that trisquel probably needs a swift kick in the ass.

incidentally, we have a script that automatically removes systemd from the trisquel live iso and spits out a fixed one, but it relies on upstart which is being abandoned by ubuntu. so while debian still has some people working to keep "not systemd" an option (if it were really optional, they would be done by now...) trisquel and ubuntu are most likely slated to have nothing in that regard. what a shame.

we honestly think that **every** user should make a machine-readable list of features they want in distros, and that this would be extremely valuable data.

on the drawing board is a **feature-schema** prototype, which in the friendliest machine-readable way possible outlines the desired and optional features of a distro such as **distro-libre**.

the key to this schema is indentation, a simulation of xml that requires zero syntax but must develop some kind of standard keywords. if everyone (we mean everyone) made a list of features they want included, this non-industry standard would be easier to develop.

distro-libre is a growing script that can automatically remaster various live isos, ensuring that people can have bootable cds and dvds with a receipt (the script) of every possible change. it is written in fig, one of the lowest-syntax, most consistent and minimal (friendly) languages in use today. you could also do distro-libre in python, but then fig translates to python.

unlike systemd, distro-libre is intended to be easily forkable. we hope that the future of remastering (and building) distros is **the application**, not the

distribution. instead of maintaining a distribution, what we would like is if you could download a program and either use it to customise a distro (with help from automation, not just by duplication of manual work) or even build one.

we expect mockery and ridicule, but instead of just talking about these things, the free media alliance offers working prototypes. the prototypes increase in sophistication over time, and would increase further with more people forking them. we encourage **collaboration between forks**, rather than worrying about setting up a large organisation (but you are welcome to do that as well.)

as a remaster tool, the way distro-libre works is not entirely new, but it works like this:

download iso -> **run automated remaster script** -> new iso

the remaster script can even download the iso for you.

the automation serves two purposes-- by default, the script is / defines the "distro" itself. instead of downloading "fig os," you download a script that produces fig os. instead of changing fig os, you change the script.

the automation that produces the default iso can also assist you in making changes. this is very basic automation, and it can be made even friendlier by moving more distro-libre logic to our indented feature-schema. that way you can still change the code and use the custom "language" (or functions) within distro-libre, but most people will use the more abstract and user friendly schema to do many of the same tasks.

in every step of the process, we encourage the use of languages and tools that are modeled after successful educational languages like logo and basic. we say "modeled after" because these aren't 1:1 duplicates, with artifacts like line numbers or type sigils-- logo has evolved and remains very low on punctuation, people use it to code without realising they are coding. that's the sort of computer language we want people to have at their fingertips.

but because these are remastering and build **applications**, there is no monopoly. if you want to fork a distro, change it entirely, you can just fork the application-- written in a language that high-schoolers and perhaps junior high-schoolers can learn to use easily enough.

we need more distros because we need more distro maintainers. obviously, the way distros are currently made lends itself to all kinds of political and organisational issues.

we do want distros to be more generic-- installers that work across more than one distro (family) like calamares and refracta installer, remaster tools that work across more than one distro (family) such as refracta tools, we even want build tools (applications) that help inexperienced users build their own distro as an educational experience (the fsf does **not** get education!) in the same way that using sbcs are an educational experience, and so on.

we need more distros-- an entire new generation of distros-- because the **current distros are gas-guzzlers**, both in terms of what they take to run and **especially** in terms of what they take to build. and it is terribly sad that the primary and original free software organisation in the world lacks the imagination or ambition for such a scheme.

we do encourage guix and hyperbola os to keep up the good work, because they are probably the most innovative distro builders that the fsf already recognises, but the old way of building distros **limits freedom** and limits opportunities for education (possibly even to fewer people than we need to keep them going, and that's a very serious problem if it's true-- do we need more evidence than gnewsense folding? if done the way we suggest, you could carry on gnewsense yourself!) and (per the charter) our job **is**:

> the free media alliance is happy to promote free software, but
>
> also welcomes thoughtful critiques of the fsfs methods and "extraneous requirements" (other than the 4 freedoms and gp

l licenses)

to create strategies for bolstering the fsf if possible, and salvaging the fsf otherwise.

we are not a monopoly: https://freemedia.neocities.org/remix-us.html we are the seed of a **free software federation**. and the gas-guzzling distros (mostly in terms of what it takes to maintain one, and the political costs and limited freedom that comes with those methods) can be phased out-- voluntarily-- with better ideas.

we are not suggesting (indeed we regularly criticise) top-down solutions like systemd, which consolidate power in the hands of even larger communities, and we are looking to make distros easier to fork, not harder.

the reason is simple-- when you take enough projects, packages, standards, even **people**-- and you put a single corporation in charge of them, you are

building a monopoly. systemd is made from projects that were easier for smaller communities or fewer developers to maintain.

by consolidating those projects first under red hat, then into systemd itself, they were lumped together (yes, we've read the nonsense that claims to refute this, it is bunk-- pure denial of something they seem most clearly aware of themselves) into something that takes a large corporation to maintain.

don't believe it? how long has it taken to "separate" back into smaller projects? if it were really modular, it wouldn't take dozens of people to work systemd back into modules. how much more obvious can that point become?

this is also, in a less sinister way, how distros themselves are created. and unlike systemd, those **were** created of necessity-- it was, once upon a time, far too much work for people to just make a "gnu/linux boot disk" and throw on whatever programs people wanted.

today that is increasingly possible, and the best direction for distros to go in. alas, it is not like egos and monopolistic attitudes do not exist in the free software community.

on the contrary-- distros want to remain distinct and are often opaque. it is the opacity, not the distinctions that are the real problem.

everyone is free to create their own free software, we are not suggesting that everyone give that up and "do it our way." all we are saying is-- if **freedom** is the real goal, let's put that freedom in the hands of the user, not just the distro maintainer. let's make distros that (like free software) are as forkable as possible, so that no user feels they are "locked-in" to theirs.

lock-in is a monopoly tactic, and has no place in free software distributions. if it is created inadvertently and there is a practical way to reduce it, then reducing it is also a good thing.

all the same, distro-libre is a simple prototype for liberating even the distros that do not participate! it is not about putting control of all distros in the hands of a large monopolistic corporation-- it is, like free software itself, about **putting control of all computing in the hands the user.** the old distros don't do that as well as they could, and it's time for an overhaul (you do you, but consider these words) of the concept itself.

a free (as in freedom) library, and federation of advocates

(borrowed from guarding and rescuing the fsf titanic)

amazon ebooks are an existential threat to libraries, and modern copyright is an existential threat to culture. for such an ominous global attack on all human culture, very little is being done about it. the free software foundation says to boycott such ebooks and drm, and that's a great place to start.

to promote free culture would be an obvious next step, since (legally and historically speaking) the threat that free culture mitigates comes from the same late-20th-century changes in law and industry that made the free software movement necessary. but the fsf sidesteps this connection, and their take on free culture is fairly condescending and surprisingly dismissive.

the sad thing is that the free culture movement has gotten off to many false starts, and while it continues to grow it does so at a glacial pace. at the same time, freedom within the absurd confines of modern european copyright is breaking down at what you might consider (in this age of climate change) to be a "glacial pace." if only there was a movement that stood against this... oh, there is.

whether the free software movement impresses you already or not, we strongly recommend making a more important issue of it. the fsf absolutely will not do so, but they have contributed their videos from libreplanet, so that is valuable. stallman's essays might work just as well if delivered as presentations at libreplanet, but those presentations bear free culture licences, and his essays do not.

copyright is the wrong tool to protect the integrity of expression, not only because it does not do so. the public domain is not a licence to misrepresent people, there are separate rules about that regardless of copyright, and fair use allows some people a way around the copyright but fair use is also a terrible (inadequate) tool for this purpose-- free culture licences are a better one.

considering all this, someone who believes that modern copyright is problematic has no sane reason to think cc by-nd is a useful licence. it basically says "you can't do anything at all with this, but we won't sue you just for having it." and the fsf actually promotes this licence.

we don't expect to change the fsf's position about this, we will simply let them have their weird, magical thinking about how despite libreplanet presentations being reasonably licensed, the fsf's webpages do not need the same benefit because of stallman's "works of opinion" shtick.

obviously he is entitled to his opinion, even if it limits the benefit people get from the fsf website, and even if "works of opinion" is blatant special pleading, and even if it misses the point of free culture. (we recommend nina paley's "rantifesto" on this topic.) stallman and the fsf are free to do that, it's simply a shame.

you could easily and most likely legally reconstruct the videos from libreplanet into a better free software website than the fsf's, although it would certainly be missing a fair amount of useful information.

the nice thing about such a website would be that it had a licence that allowed people to share the information on it with the same four freedoms that they enjoy with software. the fsf does not consider that important, but libreplanet videos mysteriously (and thankfully) give you those freedoms anyway. perhaps a future tagline for their events could be "we have no opinion!"

whinging about what the fsf won't do can only accomplish so much, and while we would recommend a petition for the fsf to stop misrepresenting free culture (with the straw man arguments and special pleading they tend to use to dismiss and argue against free culture-- not completely unlike, incredibly, the ones open source uses against free software) we also have our own solution to this:

let's do what the fsf won't.

let's create our own free software and free culture library, as we have started doing with the free media alliance: https://freemedia.neocities.org/library.html

this is intended as a node of such a library, not the library itself. we hope it will continue to grow, which is why the "donations" we ask for are not monetary-- the way you "donate" to the free media alliance is to create free works (four freedom works) or give us links to free works. as was policy for debian for some time, we do not accept

works under the gnu fdl. if you think the fdl is a good licence, or a free licence, answer these two questions:

1. why is it "more free" to restrict paper copies of a free cultural work?

2. if it is "more free" then why did wikipedia abandon the fdl as its license for articles, and why did the fsf help them do so?

works licensed under the fdl exist against a backdrop of a better-licensed wikipedia and better-licensed oer works. oer is one of the greatest success stories of free culture, and we proudly support it.

there is no licensing standard for oer, and we would suggest a "libre educational resources" (ler) standard based on our recommended licences page: https://freemedia.neocities.org/recommended-licenses.html

essentially, we would recommend (and are not the first to recommend) a standard for ler based on the four freedoms, of course. there are at least three ways to add works to our library:

1. create a useful or enjoyable work under a free licence. we may find it and add it ourselves.

2. give us a link to a useful or enjoyable work under a free licence. we may add it to our library.

3. create a free library similar to ours.

we strongly recommend the cc0 copyright waiver for your listings. this allows your library to grow without you even tending to it, as people can incorporate your listings into their own libraries. it also makes it easier for our library nodes to grow-- easier than just sending links to individual works (which you are also extremely welcome to do and we appreciate it!)

this library is not just for cultural works, but also for software. and there are many ways for these libraries to link up-- for example, if you choose to licence your listings under cc by-sa, which we do not recommend (by-sa is needlessly restrictive for a card catalog) we cannot simply fold your listing into ours, which is cc0, but we can select a few items from your list and add it to our collection. we can also (because it is at least a free culture licence) link to your library as a node.

although it is not a requirement at all, we also recommend creating an account on (and supporting, by various means) the internet archive: https://archive.org/account/signup ...it is a non-profit we are proud to promote, and a great resource if you are looking for freely-licensed works. some of what we do is merely an extension of what they have already done for years-- our

informal advocacy and promotion of the internet archive has already resulted in more contributed works and awareness.

but if you have a blog, forum, website, social media account or any other place online to talk about free software and free culture, these can be used to help expand our library and donate links to us. they can also be used to advocate for free software and free culture, whether you quote our words or use your own. let us know about this and help us expand our grassroots network across the world.

the work we are doing is not just the work of a free software organisation, but the work of librarians. librarians are the global champions of free speech and the preservation of culture. the fsf is not-- they could certainly do more in that regard. what they do for software, to be fair, is exactly what we recommend for all cultural works-- and no other movement is doing as much for free software.

if libraries alone did enough for free software, that would be wonderful. sadly, there continues to be a divide between the culture and freedom that libraries promote (regarding most of humanity's works over the entirety of written history) and the freedom that the fsf promotes-- we invite you to work with us to unite free software with libraries and libraries with free software. as with many libraries, you do not need to pay us to become a member. you only need to care and count yourself among us.

but we also invite you to help us advocate, not only by parroting our words and essays, but by contributing your own and **helping us to find solutions** to the many crises that culture faces in the 21st century. doing that on behalf of the fsf is very difficult-- some of us have tried for years! you can participate in what we do directly or indirectly, you can follow our advice or split off from us like forking an application: https://freemedia.neocities.org/remix-us.html

we even have a way for you to create your own unofficial "department" under our umbrella: https://freemedia.neocities.org/build-your-own-freedom-lab.html

the freedom lab movement was devised after the alliance was founded, and is a way to create your own miniature organisation without the bother and commitment of creating your own organisation. if you find our freedom lab concept inadequate, you could even create a freedom lab to devise better freedom labs. ideas and vision are what led to the founding of the free software movement. free expression is what enriches the physical and virtual libraries of the 20th and 21st centuries. without it, those libraries would all be diminished.

it is the absolute antithesis of libraries-- and the largest library ever built in human history-- the internet-- to try to crush free expression. for all the thoughtfulness and politeness and cooperation that serve a good purpose, humanity is a great mix of emotions and conflict and struggle. painting a "nice picture" over all interaction, and reducing the internet to such a picture, does damage to our ability to speak honestly about science, history and the problems facing the world today.

rather than destroy our libraries, we want to expand them and preserve them in the 21st century. as interest in physical libraries wanes, we hope you will help us find new purpose for them-- as well as preserve the rights needed online to bring the same freedom that librarians have always fought for, back to the internet as it is quickly becoming a place that is anything but "free as in speech."

the history of art and the history of human existence is full of rudeness, horror, ugliness, hatred, violence, exploitation and slavery. while we do not endorse these things, we do acknowledge our humanity. our libraries do not benefit from authoritarianism and sanitisation, they would only be diminished. the same is true of the internet, and we do not recommend the growing trend of sanitising all human interaction, conflating everyday speech with violence, and treating modern journalism as a hate crime or other criminal act.

the fsf says "free software, free society." we say: liberate software, liberate culture, liberate society. **it is not just about software anymore.** it is about the survival of human culture and the right to communicate with the rest of the world. these are more important things than the fsf is now capable of making them out to be. the future of free software however, is not its foundation-- but a federation. it is not just **our** federation, any more than it is just our freedom-- it is **yours**.

and the future of free culture is just getting started. please, protect our libraries. keep the internet alive, not sterile. we can do all of these things, if there are enough of us working together in our own way-- finding common ground and accepting that it is not possible to have every opinion in common, in a free society. that is no reason to not work to preserve human culture, and it helps in no small way if our computing is free. let's keep working to make software and culture more free (as in speech, as in freedom) than ever before.